(10) Python data structures

Uni of Michigan chapter 6

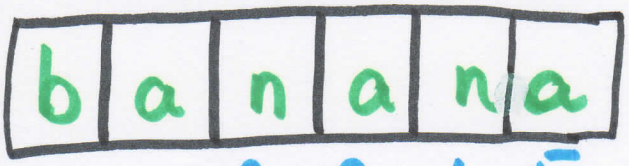
String Data Type

- Sequence of characters. It uses quotes ' or "
- + means concatenate
- When a string contains numbers, it is still a string int()
- We can convert numbers in a string into a number using

```
str1 = "Hello"
str2 = 'there'
bob = str1 + str2
print bob  Hellothere
```

```
str3 = '123'
str3 = str3 + 1   ⚠ Python error
x = int(str3) + 1
print x
124
```

Looking inside strings




```
>fruit = 'banana'
```

index value starts at 0

```
letter = fruit[1]
print letter  (a)
x = 3   w = fruit[x-1]
print w  (n)
```

```
>len(fruit)  6   (len function for length)
>print fruit[0:4]  bana   >fruit[3:]  ana   until the end
```

included — not included (up to but no including)

```
>'n' in fruit  True   >'m' in fruit  False
```

String comparison

```
word1 = 'alibaba'
word2 = 'dance'
if word1 < 'banana':
    print 'Your word comes before'
if word2 > 'banana':
    print 'Your word comes after'
```

(11) Python data structures

Uni of Michigan
chapter 6

String library

Python has function in the string library. They are built into every string. They do not modify the original string, instead they return a new string that has been altered.

```
> greet = 'Hello Steph'      > print zap > hello steph
> zap = greet.lower()        > print nop > HELLO STEPH
> nop = greet.upper()
```

To check string library list: dir(any string)

```
str.capitalize()
str.center(width[, fillchar])
str.endswith(suffix[, start[, end]])
...
```

Searching a string

find() search for a substring within a string. It finds the first occurence of the substring. If the substring is not found, it returns -1

```
> fruit = 'banana'           > print pos > 2
> pos = fruit.find('na')     > print aa > -1
> aa = fruit.find('z')  → not found
```

Search and Replace

```
> greet = 'Hello Steph'
> nst = greet.replace('Steph', 'Bob')
> print nst > Hello Bob
```

(12) # Python data structure

Uni of Michigen
chapter 6/7

Stripping white space (string)

lstrip(): remove whitespace at the lef
rstrip(): at the right
strip(): both beginning & ending

Opening files

files are located in the secondary memory.
A text file can be thought of as a sequance of lines.
open() returns a "file handle": variable used to perform operations on a file.
↳ similar to 'File -> Open' in a Word processor.

```
handle = open('filename', mode)
> handle = open('mbox.txt', 'r')
```

- returns a handle use to manipulate the file
- mode is optional 'r': read the file 'w': write to the file

What is a handle?

It is a mecanism to get out the data. It doesn't have the data however

open
read
write
close
your program

mbox.txt

From stephen.m
Return - - -
Date: - - - ...
Subject: [sakai:]

connection
It is a connection to the file's data ⚠

(13) Python Data Structure — Univ of Michigan — Chapter 7

The newline character

\n (called "newline") is a special character indicating when a line ends. One character (not two)

```
>stuff = 'Hello \n World!' > stuff 'Hello \n World!'
> print stuff  > len(stuff)
Hello                3
World!
```

- A textfile has newlines at the end of each line.

File handle as a sequence

- A file handle open for read can be treated as a sequence of strings where each line in the file is a string in the sequence.

Counting lines in a file

```
> fhand = open('mbox.txt')
> count = 0
> for line in fhand:
      count = count + 1
> print 'Line count:', count
ine count: 132045
```

① Open a file read-only
② Use a for loop to read each line
③ Count the lines and print out the number of lines

(14) Python Data Structure

Univ of Michigan
Chapter 7

Reading the whole file

```
> fhand = open("file.txt")
> inp = fhand.read()
> print len(inp)
94626
```

- We can read the whole file (newlines and all) into a single string.

Searching through a file

```
> fhand = open("file.txt")
> for line in fhand:
      line = line.rstrip()
      if line.startswith("From:"):
          print line
From: stephen.m@uct.ac.za
From: ....
```

- Each line from the file has a newline at the end. We can use rstrip() to strip the whitespace from the right. The newline is considered "whitespace" and is stripped.
- Without rstrip() we would have blank lines in between

Using in to select lines

```
> fhand = open('file.txt')
> for line in fhand:
      line = line.rstrip()
      if not '@uct.ac.za' in line:
          continue
      Print line
```

- We can look for a string anywhere in a line. We can use if not and continue to skip unecessary steps.

(15) # Python Data structures — Uni of Michigan Chapter 7/8

Prompt for file name and reject bad file name!

```
> fname = raw_input('Enter the file name: ')
> try:
      fhand = open(fname)
> except:
      print 'File cannot be opened:', fname
      exit()
> count = 0 ... (rest of the program)
```

Lists - list constants

- A list is a kind of **collection** allowing us to put many values in a **single "variable"**.
- List constants are surrounded by **square brackets** and the elements in the lists are separated by **commas**.
- A **list element** can be any **Python object** - even another list. A list can be **empty** `[]` `list()`

```
> ['red', 9, 0.17, [5,6], 'blue']
> []
> list()
```

Looking inside lists

- Just like strings, we can get any single element in a list using an index.

```
> print friends[1]
Glenn
> friends = ['Joseph', 'Glenn', 'Sally']
```

(6) # Python data structures — Uni of Michigan Chapter 8

Lists are mutable

- Strings are "immutable": we cannot change the contents of a string. We must make a new string to make any change
- List are "mutable": we can change an element of a list using the index operator.

```
> lotto = [2, 14, 26, 41, 63]      > lotto[2] = 28
> print lotto                      [2, 14, 28, 41, 63]
```

How long is a list?

len() takes a list as a parameter and returns the number of elements in the list. len() tells us the number of elements of any set or sequence (such as a string..)

Using the range function

The range function returns a list of numbers that range from 0 to one less than the parameter.

```
> print range(4)                   [0, 1, 2, 3]
> friends = ['Joseph', 'Glenn', 'Sally']
> print len(friends)               3
> print range(len(friends))        [0, 1, 2]
```

Loops

```
> for friend in friends:
      print "HNY:", friend
```

=

```
for i in range(len(friends)):
    friend = friends[i]
    print "HNY:", friend
```

(17) # Python data structures — Univ of Michigan chapter ♡

Concatenating lists using +

```
a = [1, 2, 3]  b = [4, 5, 6]  c = a + b
print c  [1, 2, 3, 4, 5, 6]
```

Building a list from scratch

. We can create an empty list and then add elements using the append method. The list stays in order and new elements are added at the end of the list.

```
> stuff = list()  > stuff.append('book')
> stuff.append(99)  > print stuff ['book', 99]
```

Is something in a List?

You can use two operators in and not in that check if an item is in a list. That returns True or False

```
> some = [1, 9, 16]  > 9 in some  True
> 15 in some  False  > 20 not in some  True
```

A list is an ordered sequence

. A list can hold many items and keeps those items in the order until we do something to change the order.
. A list can be sorted.. sort() method means "sort yourself" (unlike in strings)

```
> friends = ['Joseph', 'Glenn', 'Sally']
> friends.sort()  print friends ['Glenn', 'Joseph', ..]
>
```

(18) Python Data Structures — Uni of Michigan Chapter 8

Built-in functions and lists

Many built-in function take lists as parameter.

max(list) min(list) sum(list) len(list)

It makes some programs much simpler such as following loops:

```
> numlist = list()
> while True:
      inp = raw_input('Enter a number: ')
      if inp == 'done': break
      value = float(inp)
      numlist.append(value)
> average = sum(numlist)/len(numlist)
> print 'Average:', average
```

```
befor:
count = 0
total = 0
:
```

Best friends: strings and lists

split() breaks a string into parts and produces a list of strings. We think of these as words. We can access a particular word or loop through all the words.

```
> abc = 'With three words'
> stuff = abc.split()
> print stuff
['With', 'three', 'words']
> for w in stuff: print w
With
three
words
```

(19) Python data structures Uni of Michigan chapter 8

- When you do not specify a delimiter, multiple spaces are treated like one delimiter.
- You can specify what delimiter character to use in the splitting.

```
> line = 'A lot                    of spaces'
> etc = line.split()
> print etc ['A', 'lot', 'of', 'spaces']
> line = 'first;second;third'
> thing = line.split()
> print thing ['first;second;third'] (len(thing) 1)
> thing = line.split(';')
> print thing ['first', 'second', 'third'] (len(thing) 3)
```

The double split pattern

- We can split a line one way, and then grab one of the pieces of the line and split that piece again.

```
From stephan.marquard@uct.ac.za Sat Jan 5 09:14
words = line.split()          ① stephan.marquard@uct.ac.za
email = word[1]               ② ['stephan.marquard', 'uct.ac.za']
pieces = email.split('@')     ③ 'uct.ac.za'
print pieces[1]
```

(20) # Python data structure — Univ of Michigan Chapter 9

Dictionaries

- A dictionary is like a "bag" of value, each with its own label.
- Dictionaries are Python's most powerful data collection.
- They allow us to do fast database-like operation.
- They don't have order.
- We index the things we put in the dictionary with a "lookup tag":

```
dict() = {} empty dictionary
> purse = dict()
> purse['candy'] = 3
> purse['tissues'] = 75
> print purse
{'money': 12, 'tissues': 75, 'candy': 3}
> print purse['candy']
3
> purse['candy'] = purse['candy'] + 2
> print purse
{'money': 12, 'tissues': 75, 'candy': 5}
```

adding value & key

- Dictionaries are like lists except that they use keys instead of numbers to look up values.

Dictionary	Key	Values
purse	'money'	12
	'tissues'	75
	'candy'	5

↳ list of key

list of key: value pairs

Python data structure

Univ of Michigan chapter 9

Many counters with a dictionary

- One common use of dictionary is counting how often we see something.
- We can use the in operator to see if a key is in the dictionary
- When we encounter a new name, we need to add a new entry in the dictionary and if this is the second or later time we have seen the name, we simply add one to the count in the dictionary under that name.

```
> names = ['csev', 'cwen', 'csev', 'zqian']
> counts = dict()
> for name in names:
      if name not in counts:
          counts[name] = 1
      else:
          counts[name] = counts[name] + 1
> print counts
{'csev': 2, 'zqian': 1, 'cwen': 1}
```

Simplified counting with the get() method

common method

```
> if name in counts:
      x = counts[name]
  else:
      x = 0
```

=

Get method

```
> counts.get(name, 0)
```

0: default value if key does not exist / is not found

Python data structure

- We can use get() and provide a default value of zero when the key is not yet in the dictionary - and then just add one.

```
> counts = dict() > names = ['csev', 'cwen', 'csev', 'zqian', 'cwen']
> for name in names :
      counts[name] = counts.get(name, 0) + 1
> print counts {'csev': 2, 'zqian': 1, 'cwen': 2}
```

(default)

Counting pattern

```
> counts = dict()
> print 'Enter a line of text:'
> line = raw_input('')
> words = line.split()
> print 'Words:', words
> print 'Counting...'
> for word in words:
      counts[word] = counts.get(word, 0) + 1
> print 'Counts', counts
```

- The general pattern to count the words in a line of text is to split the line into words, then loop through the words and use a dictionary to track the count of each word independently

Definite loops and dictionaries

- Even though dictionaries are not stored in order, we can write a for loop that goes through all the entries in a dictionary - actually it goes through all the keys in the dictionary and looks up the values.

Python data structure

Uni of Michigan
chapter 9

```
> counts = {'chuck': 1, 'fred': 42, 'jan': 100}
> for key in counts:
      print key, counts[key]

jan 100
chuck 1
fred 42
```

Retrieving lists of keys and values

- You can get a list of keys, values or items (both) from a dictionary

```
> jjj = {'chuck': 1, 'fred': 42, 'jan': 100}
> print list(jjj)      ['jan', 'chuck', 'fred']
> print jjj.keys()     ['jan', 'chuck', 'fred']
> print jjj.values()   [100, 1, 42]
> print jjj.items()    [('jan', 100), ('chuck', 1),
('fred', 42)]
```

Two iteration variables!

- We loop through the key-value pairs in a dictionary using *two* iteration variables.
- Each iteration, the first variable is the key and the second variable is the corresponding value for the key.

Python data structure

Univ of Michigan
Chapter 9

```
> jjj = {'chuck': 1, 'fred': 42, 'jan': 100}
> for aaa, bbb in jjj.items():
      print aaa, bbb
```

(key) (value)

Key ① Value ②

```
jan 100
chuck 1
fred 42
```

Count the most common word in a file

```
name = raw_input("Enter file: ")
handle = open(name, 'r')
text = handle.read()
words = text.split()
counts = dict()
for word in words:
    counts[word] = counts.get(word, 0) + 1
bigcount = None  bigword = None
for word, count in counts.items():
    if bigcount is None or count > bigcount:
        bigword = word
        bigcount = count
print bigword, bigcount
```

list of words with no blanks

(25) # Python data structure

Uni of Michigan
Chapter 10

Tuples are like lists

- Tuples are another kind of sequence that functions much like a list - they have elements which are indexed starting at 0.

```
> x = ('Glenn', 'Sally', 'Joseph')      (0, 1, 2)
> print x[2]
Joseph
> for iter in x: print iter
```

But... tuples are "immutable"

- Unlike a list, once you create a tuple, you cannot alter its contents (similar to a string)

```
> z = (5, 4, 3)
> z[2] = 0
traceback
```

Things not to do with tuples

~~x.sort~~ ~~x.append~~ ~~x.reverse~~ (x tuple)

Tuples are more efficient

- Since Python does not have to build tuple structures to be modifiable, they are simpler and more efficient in terms of memory use and performance than lists.
- When we are making "temporary variables" we prefer tuples over lists.

Tuples and assignment

- We can put a tuple on the left hand side of an assignment statement
- We can even omit parentheses

```
> (x, y) = (4, 'fred')
> print y
fred
> a, b = (99, 98)
> print a
99
```

(26) # Python data structure — Univ of Michigan chapter 9

Tuples and dictionaries

- The items() method in dictionaries returns a list of (key, value) tuples

```
> d = dict > d['csev'] = 2 > d['cwen'] = 4
> for (k,v) in d.items():
      print k, v          csev 2 cwen 4
> tups = d.items()
> print tups [('csev', 2), ('cwen', 4)]
```

Tuples are comparable

- The comparison operators work with tuples. If the first item is equal, Python goes on to the next element, and so on, until it finds elements that differ.

```
> (0,1,2) < (5,1,2) True
> ('Jones', 'Sally') > ('Adams', 'Sam') True
```

Sorting lists of tuples

- We can take advantage of the ability to sort a list of tuples to get a sorted version of a dictionary.
- First we sort the dictionary by the key using the items() method.

```
> d = {'a': 10, 'b': 1, 'c': 22}  > t = d.items()
> t [('a', 10), ('c': 22), ('b', 1)]  > t.sort()
> t [('a', 10), ('b', 1), ('c', 22)]
>
```

(27) Python data structures Uni of Michigan chapter 9

Using sorted()

- We can do this even more directly using the built-in function sorted() that takes a sequence as a parameter and returns a sorted sequence.

```
> d = {'a':10, 'b':1, 'c':22}
> d.items()  [('a',10),('c',22),('b',1)]
> for k,v in sorted(d.items()):
      print k,v        a 10  b 1  c 22
```

Sort by values instead of key

- If we could construct a list of tuples of the form (value, key) we could sort by value
- We do this with a for loop that creates a list of tuples

```
> c = {'a':10, 'b':1, 'c':22}  > tmp = list()
> for k,v in c.items():
      tmp.append( (v,k) )
> print tmp   [(10,'a'),(22,'c'),(1,'b')]
> tmp.sort(reverse=True)
> print tmp  [(22,'c'),(10,'a'),(1,'b')]
```

Even shorter version

List comprehension creates a dynamic list. In this case we make a list of reversed tuples and then sort it.

```
>print sorted( [(v,k) for k,v in c.items()] )
                [(1,'b'),(10,'a'),(22,'c')]
```

Python data structures

Univ of Michigan
Chapter 9

The top 10 most common words

```
fhand = open('romeo.txt')
counts = dict()
for line in fhand:
    words = line.split()
    for word in words:
        counts[word] = counts.get(word, 0) + 1

lst = list()
for key, val in counts.items():
    lst.append( (val, key) )

lst.sort( reverse = True)
for val, key in lst[:10]:
    print key, val
```

Bonuses from worked exercise

Possible to add: `line.rstrip().upper()` (1st, 2nd) (inp being the entry from raw_inp)

Check for empty lines: `if len(inp) < 1 : break`

If problem with empty lines: `if words = [] : continue` (words being a list)

or `if len(words) < 1 : continue`

or `if line == '' : continue`